Google Summer Of Code

LabLua

Interrupt-based drivers and libraries for Ceu-Arduino

# Milestone Report 3

*Student:*
Guilherme Simas

*Mentor:*
Francisco Santanna

June 27, 2017

# Contents

# 1 Introduction

The goal of this report is to show the progress on the project from the past two weeks, comparing it with the expectation from the previous report in order to avaliate the progress made. In the first section the progress from previous report will be summarized. Following the summary, expectations for the proposed two-week period (June 23th) will be listed, followed by what was achieved.

This document concludes by discussing the next steps forward.

# 2 Previous Progress

On the previous report the ADC module was implemented in C as a learning tool ad step towards the Ceu API. It consisted on 3 functions which together enabled the user to perform analog reads in the application level, using interrupts in the backend. Function begin() starts the conversion, avaliable() returns whether or not a conversion is in progress and read() returns the result of the latest conversion.

# 3 Milestone Objective

The goal for the current two-week period is to implement a minimum functioning API in Ceu for the ADC module, based on the work done the previous milestone and other drivers currently implemented in Ceu. By the end of the 2-week period it is expected to have a sample application in Ceu using the API.

# 4 Progress

Drivers already previously implemented in Ceu were studied in order to better understand how the Ceu environment works in the context of libraries. This version of the Ceu API was implemented reutiziling the most possible of the C API. In other words, some of the functions implemented in C during the past period made their way to this API, as a means to not waste any work. The API has some apparent limitations, which will also be discussed below.

## 4.1 Ceu Environment

The Ceu language provides integration with C code. The developer is able to declare functions in C and call them from Ceu code. Furthermore, the environment also supports attachment of a block of code to a interrupt vector, in order words, the definition of interrupt service routines. By this logic, we can already port the C API to Ceu, but if we stop there we do not make use of Ceu's model. We want the application to be able to "await" the result, in order to save the cicles that are used for the conversion.

The application level should interface with the driver by merely requesting a value and awaiting its result.

The Ceu environment also provides easy integration with the values made available by the AVR libraries, as to make use of, for example, the interrupt vector numbers and the register values, the developer needs only to modify a file which declared the defined variables as "natives".

## 4.2 Ceu API

The begin() function implemented the previous period was brought to the API and names anreq(), as Analog Request. The function basely calls begin() by relaying the pin passed in as a parameter. As before, this will start the conversion after setting up the hardware.

Ceu's treatment of ISRs is done by "spawning" (spawn directive) a block code of type isr, passing in, as a parameter, the interrupt vector number. Much like the ISR in the C API, this block of code will be called automatically when the conversion is done. Unlike the previous version, however, the ISR will emit an input event to the application, carrying the value of the conversion (obtained from calling the

read() function from the C API). As such, there ir no longer any need to have a global state variable nor the available() function (which queried the state variable).

To perform an analog read, the application must call anreq() passing in the analog pin number as a parameter and subsequently, "await" the input event AN.

### 4.3 Limitations

The API expects the user to enter an await AN state before the hardware is done with the conversion. If the user does not accomplish this, the driver will still emit the event, which will be lost. It was also assumed that the user does not call anreq() again before the conversion is done. Consequences of the user doing this were not studied. Also, Ceu APIs are usually modeled as output and input events (user emits an event in order to relay a command to the API and the API emits input events to relay values and results to the user).

These limmitations are already being studied and treated as of the end of this period, but as the new API is still in an unstable state, the full implementation and description of the solutions will be left for next period.

## 5 Conclusion and steps forward

The API provides minimum functionality, leaving room for much more. As mentioned in the previous section, these concerns led to the development of a new version of the API, which as of the date of this reports is in a unstable state.

The next two-weeks period will be focused on the development of the next version of the API, which will approach the issues mentioned above.